Neural Class Expression Synthesis in $\mathcal{ALCHIQ}(\mathcal{D})^\star$

N'Dah Jean Kouagou[0000−0002−4217−897X] (✉),
Stefan Heindorf[0000−0002−4525−6865], Caglar Demir[0000−0001−8970−3850], and
Axel-Cyrille Ngonga Ngomo[0000−0001−7112−3516]

Department of Computer Science, Paderborn University, Germany
{ndah.jean.kouagou, heindorf, caglar.demir, axel.ngonga}@upb.de

Abstract. Class expression learning in description logics has long been regarded as an iterative search problem in an infinite conceptual space. Each iteration of the search process invokes a reasoner and a heuristic function. The reasoner finds the instances of the current expression, and the heuristic function computes the information gain and decides on the next step to be taken. As the size of the background knowledge base grows, search-based approaches for class expression learning become prohibitively slow. Current neural class expression synthesis (NCES) approaches investigate the use of neural networks for class expression learning in the attributive language with complement ($\mathcal{ALC}$). While they show significant improvements over search-based approaches in runtime and quality of the computed solutions, they rely on the availability of pretrained embeddings for the input knowledge base. Moreover, they are not applicable to ontologies in more expressive description logics. In this paper, we propose a novel NCES approach which extends the state of the art to the description logic $\mathcal{ALCHIQ}(\mathcal{D})$. Our extension, dubbed NCES2, comes with an improved training data generator and does not require pretrained embeddings for the input knowledge base as both the embedding model and the class expression synthesizer are trained jointly. Empirical results on benchmark datasets suggest that our approach inherits the scalability capability of current NCES instances with the additional advantage that it supports more complex learning problems. NCES2 achieves the highest performance overall when compared to search-based approaches and to its predecessor NCES. We provide our source code, datasets, and pretrained models at `https://github.com/dice-group/NCES2`

Keywords: Neural network · Description logic · Class expression learning

⋆ This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant No 860801 and the European Union's Horizon Europe research and innovation programme under the grant No 101070305. This work has also been supported by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL under the grant No NW21-059D and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation): TRR 318/1 2021 – 438445824.

1 Introduction

Class expression learning approaches [13, 14, 22, 24, 30, 35] are supervised machine learning approaches that learn class expressions in description logics: Given a knowledge base, and a subset of the individuals in the knowledge base, the goal is to learn a class expression that holds for the given individuals, i.e., describes them. For example, given the set of individuals {Marie Curie, Linus Pauling, John Bardeen, Frederick Sanger}, a learner should compute the class expression $\geq 2 hasWon.\{NobelPrize\}$ (i.e., individuals who have won at least two Nobel prizes). As the learned class expressions provide a concise and human-readable explanation for why individuals are classified as positives or negatives, class expressions can be considered explainable, interpretable white-box models. Class expression learning has important applications in ontology engineering [23], bio-medicine [25] and Industry 4.0 [4].

Although several approaches have been developed to solve class expression learning problems, most of them do not scale to large knowledge bases. In particular, approaches based on refinement operators [13, 17, 21, 22, 24, 30] and evolutionary algorithms [14] suffer from the exploration of an infinite conceptual space where each step invokes a reasoner to compute the instances of numerous intermediary refinements. Moreover, the reasoning complexity grows with the expressivity of the underlying description logic [16, 29], and existing search-based approaches cannot leverage previously solved learning problems [18].

To alleviate the aforementioned issues, Kouagou et al. [18] proposed a new family of approaches, dubbed neural class expression synthesis (NCES) approaches. These approaches work in a fashion akin to neural machine translation [9, 39] and translate (embeddings of) sets of positive/negative examples to class expressions. Extensive experiments on different datasets showed that post-training, NCES approaches do not suffer the costly exploration encountered by search-based approaches as they directly synthesize class expressions in a single forward pass in approximately one second on average. Moreover, these approaches have the ability to solve multiple learning problems at the same time as they accept batches of inputs. NCES approaches are therefore well suited for deployment in large-scale applications of class expression learning, e.g., on the web.

Despite their effectiveness, current implementations of NCES have some important limitations. First, they cannot solve learning problems beyond $\mathcal{ALC}$, e.g., data properties are not supported. Second, they assume the availability of pretrained embeddings for each input knowledge base. In this work, we propose a novel implementation of NCES that goes beyond existing instantiations in three directions: (1) We extend the supported description logic to $\mathcal{ALCHIQ}(\mathcal{D})$ to increase reasoning capabilities, (2) we improve the training data generation method, and (3) we incorporate an embedding model into the approach so that embeddings for the input knowledge base can be jointly learned with class expressions. As a result, our approach works in an end-to-end manner and does not require pretrained embeddings. Our approach achieves the highest performance overall when compared to search-based approaches and current NCES implementations.

We organize the rest of the paper as follows: First, we present related works for class expression learning and the background needed throughout the paper. Next, we describe our proposed approach and evaluate it with respect to existing state-of-the-art approaches on four benchmark datasets. Finally, we conclude the paper and introduce new directions for future work.

2 Related Work

Many approaches for class expression learning have been developed in the last decade [7, 13, 22, 23], and even recently [14, 17, 18, 32]. The former are based on refinement operators while the most recent approaches use additional techniques such as evolutionary algorithms [14] or neural networks [17, 18]. The state-of-the-art EvoLearner [14] initializes its population by random walks on the input knowledge graph where nodes are atomic classes or data values, and edges are abstract or concrete roles. The results of the random walks are then converted to description logic concepts and further refined by means of mutation and crossover operations. CELOE [23] is a state-of-the-art class expression learning algorithm tailored towards ontology engineering. It is implemented in DL-Learner [21] alongside other search-based algorithms such as OCEL [22]—which formed the basis for CELOE, and ELTL [7]—which learns concepts in the lightweight description logic $\mathcal{EL}$. ECII [32] is a search-based algorithm, too, but it does not use a refinement operator and only invokes a reasoner once for each run. CLIP [17] is an extension of CELOE that uses neural networks to predict an approximate length of the solution during concept learning. NCES (neural class expression synthesis) approaches were proposed by Kouagou et al. [18] to overcome the runtime limitations of search-based approaches. NCES instances use pretrained embeddings of input knowledge bases and translate them into class expressions in $\mathcal{ALC}$. They can solve hundreds of learning problems at a time because they accept batches of inputs.

Apart from the recent family of approaches NCES, the rest of the well-known approaches for class expression learning are search-based. Hence, these approaches need to perform numerous expensive entailment checks [29] and evaluations of candidate concepts during concept learning. Our approach inherits the scalability capability of current NCES instances and achieves superior performance on complex learning problems, i.e., learning problems involving data properties, cardinality restrictions, or inverse properties.

3 Background

In the rest of the paper, we denote a knowledge base by $\mathcal{K} = (TBox, ABox)$, i.e., a pair consisting of a terminological box, and an assertion box [18, 27]. Its sets of individuals, roles, and atomic classes are denoted by $\mathcal{N}_I$, $\mathcal{N}_R$, and $\mathcal{N}_C$, respectively. $|.|$ denotes the cardinality of a set or the size of an array. Given a non-empty one-dimensional array A and an integer i such that $0 < i \leq |A|$, both $A[i]$ and A_i denote the element of A at position i. We adopt similar notations for

Table 1. Description logic constructs supported by NCES2.

Syntax	Construct
$\mathcal{ALC}$	
r	Abstract role
$\neg C$	Negation
$C \sqcup C$	Disjunction
$C \sqcap C$	Conjunction
$\exists\ r.C$	Existential restriction
$\forall\ r.C$	Universal restriction
$\mathcal{H}$	
$r_1 \sqsubseteq r_2$	Role inclusion
$\mathcal{I}$	
r^-	Inverse role

Syntax	Construct
$\mathcal{Q}$	
$\leq n\ r.C$	Max. cardinality restriction
$\geq n\ r.C$	Min. cardinality restriction
$(\mathcal{D})$	
b	Boolean concrete role
d	Numeric concrete role
$d \leq v$	Max. numeric restriction
$d \geq v$	Min. numeric restriction
$b = \mathit{True}; b = \mathit{False}$	Boolean value restriction

high-dimensional arrays. As in [18], we convert input knowledge bases into sets of triples with RDFLib [19]. These triples are used as inputs to the embedding model in NCES2 during training, see Section 4.4.

3.1 Description Logics

Description logics [27] are a family of knowledge representation systems that are widely used in artificial intelligence, the semantic web, and automated reasoning. They are designed to express the meaning of a statement in a formal language that is then used for automated reasoning. Indeed, the web ontology language, OWL, uses description logics to represent the *TBox* of RDF ontologies. Our approach runs in $\mathcal{ALCHIQ}(\mathcal{D})$, i.e., $\mathcal{ALC}$ [33] extended with property hierarchies (used during training data generation), inverse properties, cardinality restrictions, and data properties. We present the syntax of $\mathcal{ALCHIQ}(\mathcal{D})$ in Table 1. For its semantics, we refer to Lehmann [22].

3.2 Refinement Operators

Definition 1 ([18, 24]). *Given a quasi-ordered space $(\mathcal{S}, \preceq)$, a downward (respectively upward) refinement operator on $\mathcal{S}$ is a mapping $\rho : \mathcal{S} \to 2^{\mathcal{S}}$ such that for all $C \in \mathcal{S}$, $C' \in \rho(C)$ implies $C' \preceq C$ (respectively $C \preceq C'$).*

In this work, we extend the refinement operator by Kouagou et al. [17] to the description logic $\mathcal{ALCHIQ}(\mathcal{D})$ and use the latter to generate training data for our approach, see Section 5.1 for more details.

3.3 Class Expression Learning

Definition 2 (Theoretical Solution). *Given a knowledge base $\mathcal{K}$, a target concept T, a set of positive examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$, and a set of negative examples $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to find a class expression C such that for $\mathcal{K}' = \mathcal{K} \cup \{T \equiv C\}$, we have $\forall\ e^+ \in E^+\ \forall e^- \in E^-,\ \mathcal{K}' \models C(e^+)$ and $\mathcal{K}' \not\models C(e^-)$.*

While search-based approaches such as CELOE and EvoLearner repeatedly invoke a reasoner and a heuristic function to incrementally construct the solution C, our approach directly synthesizes the solution by mapping the output (computed in approximately one second) of its neural network component to the vocabulary of tokens. In Section 4.2, we adapt Definition 2 to our approach.

3.4 Knowledge Graph Embedding

A knowledge graph is a collection of assertions. In this paper, we consider knowledge graphs $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\mathcal{E}$ is a finite set of entities and $\mathcal{R}$ is a finite set of relations. Knowledge graph embeddings are mappings of entities (and relations) into a vector space. Embeddings can be used for a variety of tasks such as link prediction [6], recommendation systems [43], and natural language processing [8]. A large number of embedding approaches have been developed in the recent past [10, 36]. They can be classified in two main categories: (1) Approaches that use only facts observed in the knowledge graph [5, 28], and (2) approaches that leverage additional available information about entities and relations, such as textual descriptions [37, 40]. Our approach uses the state-of-the-art approach ConEx [11], which belongs to the first category, as its default embedding model. We also conduct additional experiments with DistMult [41]. The results obtained with DistMult are similar to those of ConEx. They are omitted due to space constraints and can be found in our supplementary material.

3.5 The Set Transformer Architecture

Class expression learning from examples deals with set-structured inputs (see Definition 2). Several neural network architectures have been proposed to solve set-structured input tasks, the most prominent of which include Deep Set [42] and Set Transformer [20]. Deep Set encodes the elements of the input set independently and applies a pooling function, usually a summation, to represent the entire set. Contrarily, Set Transformer computes an encoding of the input set via a self-attention mechanism on its elements. In its original paper, Set Transformer outperforms other set-compatible architectures on most tasks [20], including Deep Set. For this reason, the neural network component of our approach uses the Set Transformer architecture. Its building blocks are the Multi-head Attention Block (MAB), the Set Attention Block (SAB), the Induced Set Attention Block (ISAB), and the Pooling by Multi-head Attention (PMA). Due to space constraints, we refer to [20] for more details on the Set Transformer architecture.

4 Proposed Approach (NCES2)

4.1 Preliminaries

We create a vocabulary of tokens $\mathcal{V}_\mathcal{K}$ (we simply write $\mathcal{V}$ when there is no ambiguity) for each input knowledge base $\mathcal{K}$. The vocabulary consists of all atomic concept and role names in $\mathcal{K}$ in addition to the following constructs: "$\top$" (top concept), "$\bot$" (bottom concept), "False", "True" (Boolean values), "$^-$" (for inverse properties), ":", "xsd", "double", "integer", "date" (for time data values), "$\leq$", "$\geq$", " " (white space), "." (dot), "$\sqcup$", "$\sqcap$", "$\exists$", "$\forall$", "$\neg$", "[", "]", "{", "}", "(", and ")". We add the special token "PAD" to pad all class expressions in a batch of training examples to the same length. The token also serves as the end token at inference time when parsing the output of NCES2. Finally, we add numeric data values to the vocabulary. These values are obtained by creating evenly spaced bins ranging from the lowest to the highest value observed in the knowledge base.

We now choose a fixed ordering for the elements of $\mathcal{V}_\mathcal{K}$ and use them to synthesize class expressions (more details in Section 4.6). In fact, class expressions in $\mathcal{ALCHIQ}(\mathcal{D})$ are written using tokens in $\mathcal{V}_\mathcal{K}$ as can be seen in the learning problems below, which are extracted from test datasets: LP_1 = `Man` $\sqcap$ ($\forall$ `knows.`($\neg$`SonOfGod`)) $\sqcap$ ($\leq$ `2 visitedPlace.`$\top$) (Semantic Bible), LP_2 = `Fluorine-92` $\sqcup$ `Sulfur-74` $\sqcup$ ($\exists$ `drosophila_rt.`{False}) (Carcinogenesis), LP_3 = (`Atom` $\sqcap$ (`Tin` $\sqcup$ ($\neg$`Carbon-25`))) $\sqcup$ ($\exists$ `inBond.`($\neg$`Carbon-10`)) (Mutagenesis), and LP_4 = `Measurable-Trend` $\sqcup$ ($\exists$ `related.`(`Idea` $\sqcup$ `Uprising`)) (Vicodi). We discuss the solutions computed by different class expression learning approaches for each of these learning problems in Section 5.

4.2 Learning Problem

Definition 3 (Solution by NCES2). *Given a knowledge base $\mathcal{K}$ and sets of positive/negative examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to compute a class expression C in $\mathcal{ALCHIQ}(\mathcal{D})$ (using tokens in $\mathcal{V}_\mathcal{K}$) that maximizes the F-measure and Accuracy defined by*

$$\mathbf{F}_1 = 2 \times \frac{\textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}}, \tag{1}$$

$$\textbf{Precision} = \frac{|C_I \cap E^+|}{|C_I \cap E^+| + |C_I \cap E^-|}, \ \textbf{Recall} = \frac{|C_I \cap E^+|}{|E^+|}, \tag{2}$$

$$\textbf{Accuracy} = \frac{|C_I \cap E^+| + |(\mathcal{N}_I \setminus C_I) \cap E^-|}{|E^+| + |E^-|}, \tag{3}$$

where C_I denotes the set of instances of C, and $\mathcal{N}_I$ the set of all individuals.

The metrics **Accuracy** and $\mathbf{F}_1$ are used to compare different approaches on class expression learning problems—see Table 4. One difference between Definition 2 and Definition 3 is for example that the latter targets a specific description logic

(in this case $\mathcal{ALCHIQ}(\mathcal{D})$) while the former is general, i.e., applicable to any description logic. Moreover, Definition 3 allows for approximate solutions to be returned when the exact solution is not found, while Definition 2 does not. In theory, there can be multiple solutions to the learning problem; NCES2 generates only one.

4.3 Encoding Positive and Negative Examples

Set Transformer is an encoder-decoder architecture. The encoder *Enc* consists of two ISAB layers. The decoder *Dec* is composed of one PMA layer (with $k = 1$), and a linear layer for the desired output shape. During training, the embedding model component provides embeddings for positive examples x_{pos} and negative examples x_{neg}. These two embeddings are fed to the encoder independently. The outputs are then concatenated row-wise and fed to the decoder which produces the final scores s for all tokens in the vocabulary $\mathcal{V}$:

$$O_{pos} = Enc(x_{pos}),\ O_{neg} = Enc(x_{neg}), \tag{4}$$

$$s = Dec(\text{Concat}(O_{pos}, O_{neg})). \tag{5}$$

4.4 Loss Function

Our approach is trained by minimizing two joint loss functions: (1) The loss $\mathcal{L}_1$ from the embedding model, and (2) the loss $\mathcal{L}_2$ from the class expression synthesizer. Formally, let $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ be the knowledge graph representation of the input knowledge base, and let $h \in \mathcal{E}, r \in \mathcal{R}$ be a head entity and a relation. We define $\mathcal{L}_1$ to be the binary cross-entropy loss:

$$\mathcal{L}_1(y^{hr}, \hat{y}^{hr}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} y_i^{hr} \log(\hat{y}_i^{hr}) + (1 - y_i^{hr}) \log(1 - \hat{y}_i^{hr}). \tag{6}$$

Here, $y^{hr} \in \{0,1\}^{|\mathcal{E}|}$ is the binary representation of $\{(h,r,t)|t \in \mathcal{E}\}$ in $\mathcal{G}$, i.e., $y^{hr}[\text{id}(t)] = 1$ if $(h,r,t) \in \mathcal{G}$, and $y^{hr}[\text{id}(t)] = 0$ otherwise. Accordingly, $\hat{y}^{hr} \in [0,1]^{|\mathcal{E}|}$ is the vector of scores predicted by the embedding model for all candidate tail entities. On the other hand, $\mathcal{L}_2$ is defined by

$$\mathcal{L}_2(s,t) = -\frac{1}{L} \sum_{i=1}^{L} \log \left(\frac{\exp(s_{t_i,i})}{\sum_{c=1}^{|\mathcal{V}|} \exp(s_{c,i})} \right), \tag{7}$$

where L is the maximum length of class expressions our approach can generate, $|\mathcal{V}|$ the total number of tokens in the vocabulary, $s \in \mathbb{R}^{|\mathcal{V}| \times L}$ the matrix of predicted scores for each position in the target sequence of tokens, and $t \in \{1, 2, \ldots, |\mathcal{V}|\}^L$ the vector of target token indices in the input class expression.

Our total loss $\mathcal{L}$ is defined as the average of $\mathcal{L}_1$ and $\mathcal{L}_2$, computed on the inputs $(y^{hr}, \hat{y}^{hr})$ and (s,t):

$$\mathcal{L}(y^{hr}, \hat{y}^{hr}, s, t) = \frac{\mathcal{L}_1(y^{hr}, \hat{y}^{hr}) + \mathcal{L}_2(s,t)}{2}. \tag{8}$$

During training, we alternatively sample a minibatch of N_1 training datapoints for the embedding model, and a minibatch of N_2 training datapoints for the neural synthesizer to compute $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. We then compute the gradient of $\mathcal{L}$ w.r.t. both the parameters of the embedding model and those of the synthesizer. To prevent gradient explosion and to reduce overfitting, we use gradient clipping [44], and dropout [34]. Both parameter sets are updated using the Adam [15] optimization algorithm. Note that the embeddings of positive and negative examples used by the synthesizer—see Section 4.3—come from the embedding model and are hence dynamically updated during training. This way, we are able to learn embeddings that are not only finetuned for class expression learning, but also faithful to the input background knowledge.

4.5 Measuring Performance During Training

In our previous work [18], we introduced two metrics to quantify the performance of neural synthesizers during training[1]. We use the same metrics in this work. The first metric is called "*Soft Accuracy*" and is equivalent to the *Jaccard index* between the set of predicted tokens and the set of true tokens in the input expression. The second metric is called "*Hard Accuracy*", and compares the tokens in the prediction and target expressions position-wise, i.e., taking into account their order of appearance. We refer to [18] for the mathematical expressions of these metrics.

4.6 Class Expression Synthesis

We synthesize class expressions by mapping the output scores s (see Equation 5) to the vocabulary. Specifically, we select the highest-scoring token in the vocabulary for each position i along the sequence dimension:

$$\mathrm{id}_i = \underset{c \in \{1,\ldots,|\mathcal{V}|\}}{\arg\max}\ s_{c,i}, \tag{9}$$

$$\texttt{synthesized\_token}_i = \mathcal{V}[\mathrm{id}_i]. \tag{10}$$

The predicted tokens are concatenated to construct a class expression. Note that we ignore all tokens appearing after the special token "PAD".

4.7 Model Ensembling

Several works have highlighted that combining different neural models trained even on the same dataset usually performs better than each individual model [12, 31]. This technique is known as model ensembling. In this work, we trained three instances of our approach, NCES2, which all use the Set Transformer architecture

[1] These metrics are only used during training. When comparing NCES2 to state-of-the-art approaches on class expression learning on the test sets, we use metrics based on the number of covered/ruled-out positive/negative examples for all approaches.

but with different numbers of inducing points: $m = 32$, $m = 64$, and $m = 128$. We compute ensemble predictions by averaging the predicted scores for each token post training. Overall, we consider four ensemble models: NCES2$_{m=\{32,64\}}$, NCES2$_{m=\{32,128\}}$, NCES2$_{m=\{64,128\}}$, and NCES2$_{m=\{32,64,128\}}$. A class expression is then synthesized as described in Section 4.6 using the average scores.

5 Evaluation

5.1 Experimental Setup

Datasets We used four benchmark datasets in our experiments: Vicodi [26], Carcinogenesis [38], Mutagenesis [38], and Semantic Bible[2]. The Carcinogenesis and Mutagenesis knowledge bases describe chemical compounds and how they relate to each other. The Semantic Bible knowledge base describes the New Testament, and the Vicodi knowledge base describes the European history. We summarize the statistics of each dataset in Table 2.

Training Data Generation Training NCES2 requires numerous class expressions with their sets of positive and negative examples[3]. To this end, we extend the refinement operator by Kouagou et al. [17] to the description logic $\mathcal{ALCHIQ}(\mathcal{D})$ so that we can generate all forms of class expressions supported by NCES2 (see Table 1). Moreover, we improve upon the training data generation method used in [18]. Since most knowledge bases contain thousands to millions of individuals, current NCES approaches subsample the initial sets of positive/negative examples for each learning problem in the training set. This technique is inefficient because only a few examples are seen during training which results in poor performance on learning problems where, e.g., a different random seed is used to construct the sets of examples. To alleviate this issue, we construct multiple copies (2 copies in our experiments) of a given learning problem and assign different subsets of examples to each copy. The clear advantage of this new sampling technique is that it allows each learning problem to be seen from different perspectives and hence better understood. Note that this sampling technique is only applied to the training set. The statistics of the generated data are given in Table 2.

Hyper-parameter Search Following [18], we employ a random search [2] to find the best hyper-parameter values for NCES2. Specifically, we find highly performing values on one dataset (we used Carcinogenesis for this purpose) and use them on the rest of the datasets. Note that the total number of examples n is an exception as it depends on the size of $\mathcal{N}_I$, i.e., the total number of individuals in the given knowledge base. Nonetheless, we used the same formula to compute the optimal value for n : $\min\left(\frac{|\mathcal{N}_I|}{2}, 1000\right)$. The selected values for hyper-parameters are presented in Table 3.

[2] `https://www.semanticbible.com/ntn/ntn-overview.html`

[3] Positive examples are instances of the class expression while negative examples are the rest of the individuals in $\mathcal{N}_I$.

Table 2. Statistics of the benchmark datasets. In the table, we use the following notations and abbreviations: Set of individuals ($\mathcal{N}_I$), set of atomic classes ($\mathcal{N}_C$), object properties (*Obj. Pr.*), data properties (*D. Pr.*), vocabulary ($\mathcal{V}$), and learning problems in the test set (LPs).

Dataset	$\lvert\mathcal{N}_I\rvert$	$\lvert\mathcal{N}_C\rvert$	$\lvert Obj.\ Pr.\rvert$	$\lvert D.\ Pr.\rvert$	$\lvert TBox\rvert$	$\lvert ABox\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert Train\rvert$	$\lvert LPs\rvert$
Carcinogenesis	22,372	142	4	15	144	74,223	198	19,635	100
Mutagenesis	14,145	86	5	6	82	47,722	133	9,705	100
Semantic Bible	724	48	29	9	56	3,106	125	11,069	100
Vicodi	33,238	194	10	2	204	116,181	242	46,094	100

Table 3. Hyper-parameter settings per dataset. L is the maximum length of expressions synthesized by NCES2, lr the learning rate, N_1 the minibatch size for the embedding model, N_2 the minibatch size for the synthesizer, n the number of (positive and negative) examples, d the embedding dimension, gc the gradient clipping value.

Dataset	*epochs*	*optimizer*	lr	d	N_1	N_2	L	n	gc
Carcinogenesis	200	Adam	0.001	50	1,024	512	48	1,000	5
Mutagenesis	200	Adam	0.001	50	1,024	512	48	1,000	5
Semantic Bible	200	Adam	0.001	50	1,024	512	48	362	5
Vicodi	200	Adam	0.001	50	1,024	512	48	1,000	5

Hardware We trained NCES2 using 24GB RAM, 16 AMD EPYC 7713 CPUs @3.10GHz, and a single NVIDIA RTX A5000 GPU with 24GB memory. Because search-based approaches do not support GPU computation, we conduct experiments on class expression learning on the test sets (see Table 4) using a server with 16 Intel Xeon E5-2695 CPUs @2.30GHz and 128GB RAM. Due to space constraints, we report the number of parameters and the training time in our supplementary material.[4]

5.2 Results and Discussion

Training Curves NCES2 was trained for 200 epochs on each dataset. Training curves are shown in Figure 1. From the figure, we can observe that NCES2 is able to accurately map instance data (positive/negative examples) to the corresponding class expressions on the training set. This is witnessed by a performance of over 95% *Hard Accuracy* (recall the definition in Section 4.5) on all datasets. In addition, the convergence rates are higher on the largest datasets (Carcinogenesis and Vicodi), which suggests that NCES2 learns faster on large datasets. Similar observations hold for the *Soft Accuracy* curves which we do not report due to space constraints; in particular, the *Soft Accuracy* is over 95% on all datasets. The training curves for this metric can be found in our supplementary material. The exact values during training can be found in our repository[5].

[4] `https://github.com/dice-group/NCES2/blob/main/supplement_material.pdf`
[5] `https://github.com/dice-group/NCES2`

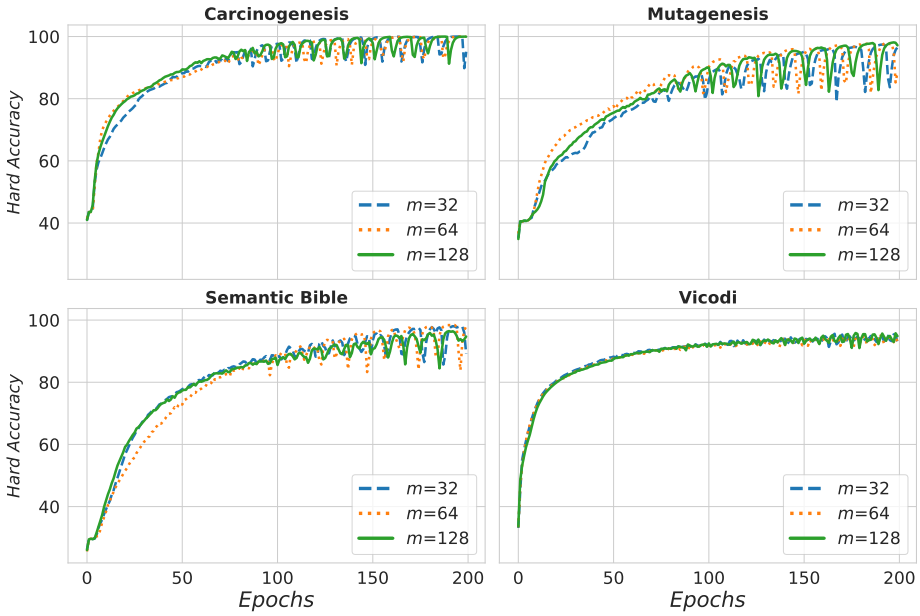

Fig. 1. Training accuracy curves using the ConEx embedding model. m is the number of inducing points.

Comparison to Baseline Approaches

NCES2 vs. Search-based Approaches We ran extensive experiments comparing NCES2 to the search-based approaches EvoLearner, CELOE, ELTL, and ECII. The results are presented in Table 4. As done in [14, 18], we employ a timeout of 5 minutes per approach on each learning problem. ECII and ELTL do not support a timeout configuration and were therefore executed with their default settings. On the one side, the results in Table 4 suggest that NCES2 significantly outperforms search-based approaches in runtime on all datasets as it synthesizes a solution in less than a second on average. The standard deviation of NCES2's prediction runtime is zero because it computes solutions for all learning problems at the same time as a single forward pass of a batch of inputs. This leads to the same prediction time for all learning problems and therefore a zero standard deviation. We employed the Wilcoxon Rank Sum Test to check for performance difference significance. The significance level is 5% and the null hypothesis that the compared quantities share the same distribution. We also achieve better performance in terms of F-measure on large datasets (Carcinogenesis and Vicodi) while remaining the second best on the Mutagenesis dataset with NCES2$_{m=\{32,128\}}$ behind EvoLearner. Meanwhile, we observe a poor performance on the Semantic Bible dataset with an average F-measure of 79.40% (NCES2$_{m=\{32,64\}}$) compared to 88.60% for CELOE. We attribute this to the data hunger of deep learning

Table 4. Evaluation results per dataset. The star (*) indicates statistically significant differences between the best search-based and the best synthesis-based approaches. Underlined values are the second best. Here, NCES2 uses the embedding model ConEx.

	F_1 (%)			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	29.24±39.22	74.46±37.59	**88.60***±19.50	22.63±35.21
EvoLearner	89.34±15.80	**95.37**± 8.02	<u>88.38</u>±12.50	76.99±26.37
ELTL	14.46±28.48	36.33±34.98	35.21±31.74	8.58±22.94
ECII	18.91±31.46	34.33±31.53	32.79±32.18	29.20±30.81
$NCES2_{m=32}$	83.56±33.11	76.79±38.61	70.77±33.73	82.36±32.05
$NCES2_{m=64}$	83.92±33.16	78.25±37.18	71.77±34.03	82.64±31.28
$NCES2_{m=128}$	86.06±32.63	73.21±38.31	69.95±36.13	83.50±30.85
$NCES2_{m=\{32,64\}}$	<u>92.13</u>±24.61	83.09±34.04	79.40±32.22	<u>90.67</u>±24.07
$NCES2_{m=\{32,128\}}$	91.01±26.80	<u>86.33</u>±31.54	77.80±34.52	87.68±26.48
$NCES2_{m=\{64,128\}}$	**92.57***±24.08	84.18±32.30	78.92±32.84	86.49±29.33
$NCES2_{m=\{32,64,128\}}$	91.29±24.96	85.12±32.28	77.00±35.25	**91.06***±23.97
	Accuracy (%)			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	62.96±22.56	87.33±17.80	95.85± 8.71	78.59±15.83
EvoLearner	**99.68***± 0.81	**99.52***± 2.17	**97.43**± 4.74	**97.79***± 6.74
ELTL	19.37±32.31	40.58±35.33	39.40±29.92	41.67±44.29
ECII	27.17±38.40	32.52±33.31	29.06±33.37	71.05±39.43
$NCES2_{m=32}$	93.86±19.54	88.77±25.13	88.03±20.40	93.87±21.39
$NCES2_{m=64}$	95.16±17.13	88.95±26.21	85.89±24.03	95.21±18.43
$NCES2_{m=128}$	92.56±22.01	89.09±24.20	88.99±20.81	96.29±16.11
$NCES2_{m=\{32,64\}}$	95.64±18.42	90.75±24.83	88.32±24.55	96.04±17.30
$NCES2_{m=\{32,128\}}$	95.15±16.76	<u>92.22</u>±22.65	<u>91.13</u>±20.23	95.93±17.39
$NCES2_{m=\{64,128\}}$	<u>95.77</u>±17.03	90.81±23.69	90.45±21.98	95.78±18.67
$NCES2_{m=\{32,64,128\}}$	95.39±18.33	90.55±24.19	88.99±24.11	<u>96.43</u>±17.01
	Runtime (sec.)			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	268.90±116.04	165.27±145.11	172.04±140.27	334.99±43.87
EvoLearner	62.21±26.11	70.77±47.53	18.44±5.53	236.92±80.90
ELTL	26.15±2.11	15.83±16.56	4.73±0.98	335.90±205.39
ECII	25.62±6.11	20.40±4.00	6.73±1.67	37.12±25.12
$NCES2_{m=32}$	**0.02***±0.00	**0.02***±0.00	**0.01***±0.00	**0.03***±0.00
$NCES2_{m=64}$	<u>0.03</u>±0.00	<u>0.03</u>±0.00	<u>0.01</u>±0.00	<u>0.03</u>±0.00
$NCES2_{m=128}$	0.03±0.00	0.03±0.00	0.02±0.00	0.04±0.00
$NCES2_{m=\{32,64\}}$	0.05±0.00	0.05±0.00	0.03±0.00	0.06±0.00
$NCES2_{m=\{32,128\}}$	0.06±0.00	0.06±0.00	0.03±0.00	0.06±0.00
$NCES2_{m=\{64,128\}}$	0.06±0.00	0.06±0.00	0.03±0.00	0.07±0.00
$NCES2_{m=\{32,64,128\}}$	0.09±0.00	0.09±0.00	0.05±0.00	0.10±0.00

Table 5. Solution per approach for learning problems LP_1, LP_2, LP_3, and LP_4 presented in Section 4.1. We consider the three best approaches NCES2, CELOE, and EvoLearner.

	Prediction	**F$_1$** (%)
	CELOE	
LP_1	`Man`	99.70
LP_2	¬Bond	0.72
LP_3	`Bond` ⊔ (`Atom` ⊓ (¬`Carbon-25`))	99.79
LP_4	`Flavour` ⊔ (¬`War`)	1.14
	EvoLearner	
LP_1	`Man`	99.70
LP_2	`Sulfur-74` ⊔ (∃ `drosophila_slrl`.{True})	83.33
LP_3	`Atom` ⊔ (∃ `inBond.Atom`)	99.78
LP_4	`Intellectual-Construct`	92.59
	NCES2	
LP_1	`Man`	99.70
LP_2	`Fluorine-92` ⊔ `Sulfur-74` ⊔ (∃ `drosophila_rt`.{False})	100.00
LP_3	(`Atom` ⊓ (`Oxygen-45` ⊔ (¬`Oxygen`))) ⊔ (∃ `inBond`.(¬`Carbon-10`))	97.10
LP_4	`Measurable-Trend` ⊔ (∃ `related`.(`Idea` ⊔ `Uprising`))	100.00

models since Semantic Bible is the smallest dataset with only 724 individuals and 48 atomic classes (cf. Table 2).

On the other side, Table 5 presents the predictive performance of the three best approaches NCES2, EvoLearner, and CELOE on the learning problems introduced in Section 4.1. NCES2 outperforms its competitors in F-measure on LP_2 and LP_4 as it computes the exact solutions for these learning problems. CELOE fails to find suitable solutions and achieves 0.72% and 1.14% F-measure on LP_2 and LP_4, respectively. EvoLearner computes approximate solutions with 83.33% and 92.59% F-measure for LP_2 and LP_4, respectively. All three approaches achieve comparable performance on LP_1 and LP_3.

The effectiveness of NCES2 is demonstrated by its ability to compute (i.e. synthesize) expressions it has never seen during training, e.g., LP_2 and LP_4. We hence believe that NCES2 should serve as a robust alternative on large knowledge bases where search-based approaches are prohibitively slow.

NCES2 vs. NCES To quantify the main differences between NCES2 and current NCES approaches, we compare them on the test sets (the 100 unseen learning problems on each knowledge base). Some of these learning problems have solutions in $\mathcal{ALC}$ while others can only be solved in $\mathcal{ALCHIQ}(\mathcal{D})$. Both approaches use the ConEx embedding model, and the Set Transformer architecture with 32 inducing points as the synthesizer. The results given by the two approaches are reported in Table 6. From the table, we can observe that NCES2 significantly outperforms NCES on all datasets with an absolute difference of up to 32.08% F-measure on the Vicodi dataset. These large differences in performance show the superiority of

Table 6. Comparison of NCES2 and NCES on test datasets. NCES2$^{✂}$ and NCES2$^{\mathcal{ALC}}$ are ablations of NCES2. The first ablation corresponds to NCES2 without the improved data generator. The second ablation corresponds to NCES2 trained on the same data as NCES, i.e., data in $\mathcal{ALC}$ on which we apply the improved data generator. All approaches use 32 inducing points and the ConEx embedding model.

	$\mathbf{F}_1$ (%)			
	NCES2	NCES2$^{✂}$	NCES2$^{\mathcal{ALC}}$	NCES
Carcinogenesis	**83.56*** ± 33.11	78.52 ± 36.72	71.24 ± 38.43	67.86 ± 41.47
Mutagenesis	**76.79*** ± 38.61	52.70 ± 44.07	53.08 ± 43.78	68.20 ± 41.70
Semantic Bible	**70.77*** ± 33.73	66.33 ± 37.02	64.33 ± 37.15	63.61 ± 35.60
Vicodi	**82.36*** ± 32.05	75.86 ± 34.50	52.57 ± 40.85	50.28 ± 43.68

NCES2 over NCES; in particular, they reveal the impact of the improved training data generator and the expressiveness of $\mathcal{ALCHIQ}(\mathcal{D})$ as an ablation of any of these leads to a decrease in performance, see the results achieved by NCES2$^{✂}$ and NCES2$^{\mathcal{ALC}}$ in Table 6. Nevertheless, the two approaches have comparable prediction time. NCES2 should therefore be preferred over NCES on most class expression learning tasks.

6 Conclusion and Future Work

We proposed an extension of NCES, a recent and scalable approach for class expression learning in $\mathcal{ALC}$. Our approach is called NCES2 and it supports the description logic $\mathcal{ALCHIQ}(\mathcal{D})$. NCES2 encodes positive and negative examples into real-valued vectors via its embedding model component, and uses a Set Transformer model to synthesize class expressions. This way, we reduce the expensive task of class expression learning, which is usually regarded as a search problem in an infinite conceptual space, to additions and multiplications in vector spaces. Our experiments demonstrated that NCES2 significantly outperforms all search-based approaches in runtime while synthesizing high-quality solutions for most learning problems. Moreover, NCES2 outperforms current NCES instances w.r.t. the F-measure of the computed solutions. NCES2 should therefore serve as a strong alternative to existing approaches especially when many learning problems are to be solved on a single knowledge base.

Currently, the training data generator in NCES2 uses an OWL reasoner to compute instances of the generated expressions. This is impractical for large knowledge bases such as DBpedia [1]. In the future, we will investigate ways to replace the reasoner by a SPARQL query-based method, such as the one by Bin et al. [3], to scale NCES2 to large datasets.

Bibliography

[1] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735, Springer (2007)

[2] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)

[3] Bin, S., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.C.: Towards sparql-based induction for large-scale rdf data sets. In: ECAI 2016, pp. 1551–1552, IOS Press (2016)

[4] Bin, S., Westphal, P., Lehmann, J., Ngonga, A.: Implementing scalable structured machine learning for big data in the sake project. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 1400–1407, IEEE (2017)

[5] Bordes, A., Glorot, X., Weston, J., Bengio, Y.: A semantic matching energy function for learning with multi-relational data - application to word-sense disambiguation. Mach. Learn. **94**(2), 233–259 (2014)

[6] Bordes, A., Usunier, N., García-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS, pp. 2787–2795 (2013)

[7] Bühmann, L., Lehmann, J., Westphal, P.: Dl-learner—a framework for inductive learning on the semantic web. Journal of Web Semantics **39**, 15–24 (2016)

[8] Chen, M., Zaniolo, C.: Learning multi-faceted knowledge graph embeddings for natural language processing. In: IJCAI, pp. 5169–5170 (2017)

[9] Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. In: SSST@EMNLP, pp. 103–111, Association for Computational Linguistics (2014)

[10] Dai, Y., Wang, S., Xiong, N.N., Guo, W.: A survey on knowledge graph embedding: Approaches, applications and benchmarks. Electronics **9**(5), 750 (2020)

[11] Demir, C., Ngomo, A.N.: Convolutional complex knowledge graph embeddings. In: ESWC, Lecture Notes in Computer Science, vol. 12731, pp. 409–424, Springer (2021)

[12] Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q.: A survey on ensemble learning. Frontiers of Computer Science **14**(2), 241–258 (2020)

[13] Fanizzi, N., d'Amato, C., Esposito, F.: DL-FOIL concept learning in description logics. In: ILP, Lecture Notes in Computer Science, vol. 5194, pp. 107–121, Springer (2008)

[14] Heindorf, S., Blübaum, L., Düsterhus, N., Werner, T., Golani, V.N., Demir, C., Ngomo, A.N.: Evolearner: Learning description logics with evolutionary algorithms. In: WWW, pp. 818–828, ACM (2022)

[15] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[16] Konev, B., Ozaki, A., Wolter, F.: A model for learning description logic ontologies based on exact learning. In: AAAI, pp. 1008–1015, AAAI Press (2016)
[17] Kouagou, N.J., Heindorf, S., Demir, C., Ngomo, A.N.: Learning concept lengths accelerates concept learning in ALC. In: ESWC, Lecture Notes in Computer Science, vol. 13261, pp. 236–252, Springer (2022)
[18] Kouagou, N.J., Heindorf, S., Demir, C., Ngonga Ngomo, A.C.: Neural class expression synthesis. In: European Semantic Web Conference, pp. 209–226, Springer (2023)
[19] Krech, D.: Rdflib: A python library for working with rdf. Online https://github. com/RDFLib/rdflib (2006)
[20] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set transformer: A framework for attention-based permutation-invariant neural networks. In: International conference on machine learning, pp. 3744–3753, PMLR (2019)
[21] Lehmann, J.: Dl-learner: learning concepts in description logics. The Journal of Machine Learning Research (2009)
[22] Lehmann, J.: Learning OWL class expressions, vol. 22. IOS Press (2010)
[23] Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. Journal of Web Semantics (2011)
[24] Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Machine Learning **78** (2010)
[25] Lehmann, J., Völker, J.: Perspectives on ontology learning, vol. 18. IOS Press (2014)
[26] Nagypál, G.: History ontology building: The technical view. Humanities, Computers and Cultural Heritage p. 207 (2005)
[27] Nardi, D., Brachman, R.J., et al.: An introduction to description logics. Description logic handbook **1** (2003)
[28] Nickel, M., Tresp, V., Kriegel, H.: Factorizing yago: scalable machine learning for linked data. In: Proc. of WWW (2012)
[29] Ozaki, A.: Learning description logic ontologies: Five approaches. where do they stand? KI-Künstliche Intelligenz (2020)
[30] Rizzo, G., Fanizzi, N., d'Amato, C.: Class expression induction as concept space exploration: From dl-foil to dl-focl. Future Generation Computer Systems (2020)
[31] Sagi, O., Rokach, L.: Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(4), e1249 (2018)
[32] Sarker, M.K., Hitzler, P.: Efficient concept induction for description logics. In: Proc. of AAAI (2019)
[33] Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial intelligence pp. 1–26 (1991)
[34] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
[35] Tran, T.L., Ha, Q.T., Hoang, T.L.G., Nguyen, L.A., Nguyen, H.S.: Bisimulation-based concept learning in description logics. Fundamenta Informaticae **133**(2-3), 287–303 (2014)

[36] Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering (2017)
[37] Wang, Z., Li, J., LIU, Z., TANG, J.: Text-enhanced representation learning for knowledge graph. In: Proc. of IJCAI (2016)
[38] Westphal, P., Bühmann, L., Bin, S., Jabeen, H., Lehmann, J.: Sml-bench–a benchmarking framework for structured machine learning. Semantic Web **10**(2), 231–245 (2019)
[39] Wu, Y., Schuster, M., Chen, Z., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)
[40] Xie, R., Liu, Z., Jia, J., Luan, H., Sun, M.: Representation learning of knowledge graphs with entity descriptions. In: Proc. of AAAI (2016)
[41] Yang, B., Yih, S.W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: Proceedings of the International Conference on Learning Representations (ICLR) 2015 (May 2015)
[42] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. Advances in neural information processing systems **30** (2017)
[43] Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y.: Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 353–362 (2016)
[44] Zhang, J., He, T., Sra, S., Jadbabaie, A.: Why gradient clipping accelerates training: A theoretical justification for adaptivity. arXiv preprint arXiv:1905.11881 (2019)